Jack Johanneson

11/26/23

IT FDN 110

Assignment07

Constructors and Inheritance

Introduction

Following the theme of last week, the tools introduced this week focus on increasing the amount of reusable code in our programs. Reducing bulk, again, makes our code more readable and more easily edited. If code is rewritten several times you will have to edit every iteration in series. Constructors help us instantiate commonly used classes, other tools introduced like setters and getters allow us to more closely monitor the accessibility and changes made to variables within these commonly used classes. To further nest these tools we can employ inheritance to reduce bulk within our similar classes.

Constructors

A constructor is used to initialize an object. In the case of this week we created a class called 'Person', all people have names, so these are provided upon the instantiation of the person object. This means that it is not necessary to assign these variables manually every time a Person object is created.

```
class Person:

    def __init__(self, first_name: str, last_name: str):
        self._first_name = first_name
        self._last_name = last_name
```

Figure 1. The 'Person' class uses a constructor to assign the first and last name variables.

Inheritance

Some classes will fall within the umbrella of another class, this can generate redundant code. For example, a person has a name but so does a student, professor, TA, janitor, etc. These different roles will have varying attributes to track, but some will be shared. Inheritance allows us to distribute code commonly needed by these roles, so our code is less repetitive. To illustrate this in our assignment this week we created a 'Student' class which inherits attributes from the 'Person' class previously described. When a student is created a person and their associated initialization code is ran, along with any code specific to 'Student' class.

```python
class Student(Person):

    def __init__(self, first_name: str, last_name: str, course: str):
        super().__init__(first_name, last_name)
        self._course = course
```

Figure 2 Student uses super() to use the constructor of 'Person' to assign the am of the student, and the student specific 'course' variable is assigned afterward.

Summary

While initially daunting, these are powerful tools that when properly utilized will make programs more easily managed. The goal of making our code as modular as possible will make it more approachable to other people reading our code and more easily debugged because issues are more likely to come from a single point rather than a mis-edited region of repeated code. The privacy restriction and access allowed in between classes and functions is important to track here as references can easily become messy.